Proceedings Paper:
Law, Timothy R., Hancox, Jonny, Cheng, Tammy M.K. et al. (4 more authors) (2016) Optimisation of a Molecular Dynamics Simulation of Chromosome Condensation. In: Proceedings - 28th IEEE International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2016. 28th IEEE International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2016, 26-28 Oct 2016 IEEE Computer Society , USA , pp. 126-133.

https://doi.org/10.1109/SBAC-PAD.2016.24

Optimisation of a Molecular Dynamics Simulation of Chromosome Condensation

Timothy R. Law , Jonny Hancox[y], Tammy M. K. Cheng[z], Raphaël A. G. Chaleil[z],
Steven A. Wright , Paul A. Bates[z] and Stephen A. Jarvis
Department of Computer Science, University of Warwick, Coventry, UK
[y]*Health and Life Sciences Team, Intel Corporation, St. Clare House, London, UK*
[z]*Biomolecular Modelling Laboratory, The Francis Crick Institute, London, UK*
Email: timothy.law@warwick.ac.uk

Abstract—We present optimisations applied to a bespoke biophysical molecular dynamics simulation designed to investigate chromosome condensation. Our primary focus is on domain-specific algorithmic improvements to determining short-range interaction forces between particles, as certain qualities of the simulation render traditional methods less effective. We implement tuned versions of the code for both traditional CPU architectures and the modern many-core architecture found in the Intel Xeon Phi coprocessor and compare their effectiveness. We achieve speed-ups starting at a factor of 10 over the original code, facilitating more detailed and larger-scale experiments.

I. Introduction

Genomes can be large–the DNA comprising the human genome is approximately 2m long when stretched out. In order to fit inside the nucleus of each cell, the DNA is wrapped around many bundles of proteins to form packaging units called nucleosomes, collectively comprising a structure called chromatin. Chromatin's structure varies with the cell cycle; when the time comes for the cell to divide, it moves from the loose conformation of nucleosomes and DNA linkers (often likened to "beads on a string") to a more tightly compacted version typically associated with chromosomes. This process is known as *chromosome condensation*, and it is thought to be affected by protein complexes known as condensins, although exactly how it happens is currently not well understood.

Current research looks to leverage computational techniques to answer this question. In this paper we discuss optimisations applied to one such biophysical molecular dynamics simulation, developed recently by Cheng et al. [1]. The simulation is used to study the effects of different models of condensin interaction on the condensation of conformations of nucleosomes and linker DNA determined via *in vitro* methods (see Figure 1). In this paper we present optimisations that make it feasible to run the lengthy simulations required.

The contributions presented in this paper are:

- Development of *projection sorting*, an improvement to the computation of short-range interaction forces between particles under certain organisational conditions;
- Analysis and implementation of the most effective thread and vector parallelism strategies for both the above and for other kernels in this simulation, leading to overall speedups starting at 10 and increasing with dataset size;
- Investigation of the resulting performance on the Intel Xeon and Xeon Phi platforms, and discussion of characteristics that make some kernels more suitable for execution on the coprocessor.

Section II discusses related work and provides a brief overview of the simulation and its component parts. In Section III we discuss the computation of repulsion forces, our algorithm for computing them, and important implementation details. Similarly, Section IV deals with condensin binding site interactions. Section V presents an analysis of the resulting performance and the work concludes in Section VI.

II. Background

A. Related work

Computational simulations are widely used in the life sciences, spanning a variety of domains of investigation from protein structures to cell pathways, with numerous software packages available. Perhaps the most well known molecular dynamics program applicable to biological research is NAMD [2], which has been heavily optimised for a variety of highly parallel systems over the last 20 years [3], [4], [5]. Other well known molecular dynamics codes include LAMMPS [6], DL_POLY [7] and GROMACS [8].

As clock-speeds drop off, exploiting the increasing amounts of available parallelism at the vector and on-chip levels becomes more and more important. Modern many-core architectures such as Intel's Knights Corner (KNC) and the upcoming Knights Landing, and Nvidia's Kepler and Maxwell GPU architectures, demand much more from implementations in order to extract maximum performance. Significant work has gone into optimising molecular dynamics applications for such architectures [9], [10], [11].

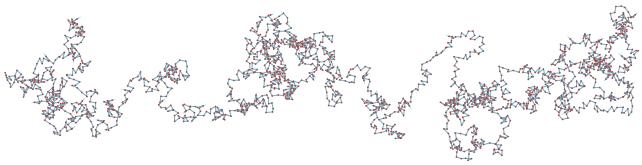

Fig. 1: A visualisation of one of the datasets used—a conformation of yeast DNA dotted with nucleosomes.

The primary focus of this paper is optimisation of a pairwise short-range force calculation, where interactions take place between all pairs of particles within a *cut-off radius*, r_c, of each other. Owing to their computational expense and widespread applicability, forces such as these have received significant attention in the MD literature. Many modern MD codes, including NAMD and LAMMPS, use a combination of *cell lists* [12], [13] and *Verlet lists* [14].

Cell lists are constructed by discretising the simulation space into cubic partitions and binning particles according to the partition they reside in. This reduces the number of distance checks that must be performed to only a small neighbourhood of partitions. This was the approach used by Cheng et al. [1].

The cell lists can further be used to cheaply construct a Verlet list for each particle, containing all other particles within distance $r_v = r_c + kr_s$, where r_c is the cut-off radius for the force, outside which its magnitude is zero, and r_s is the *skin distance* of the simulation, chosen such that no particle may move further in a single timestep. The Verlet lists may then be used to efficiently calculate forces for up to k timesteps, after which the lists must be rebuilt. k should be chosen to strike a balance between increasing the size of the Verlet lists, and decreasing the frequency of the rebuilds.

Another class of optimisations involves reordering stored particles such that those local to each other in three dimensional simulation space are also local in the computer's memory (a one dimensional space). Spatial locality is important in contemporary computer architectures, whose performance often depends on being able to work around memory latency by means of reuse within multiple layers of cache, and the ability to predict and prefetch data likely to be needed in the near future. Yao et al. discuss sorting particles along an axis of the simulation domain [15]. Anderson et al. demonstrate a successful application of a more sophisticated approach, whereby particles are ordered according to their distance along a space-filling Hilbert curve [9]. The Hilbert curve is chosen due to its locality preserving properties [16].

This simulation has the advantage that, due to the tension forces enacted by DNA linkers between each nucleosome, two nucleosomes that are next to each other in memory can *never* stray far from each other; this is in contrast to liquid or gas MD simulations. The string can however wrap around on itself leading to interactions between distant nucleosomes.

B. Simulation

The simulation treats the nucleosomes as uniform particles in a molecular dynamics simulation, with radius 5nm. The DNA linkers are modelled as ideal springs following Hooke's law, connecting each nucleosome to the next. Nucleosomes are free to move according to Brownian motion, subject to certain constraints:

Tension forces exerted by DNA linkers (modelled as ideal springs);
Repulsion forces between nucleosomes that are intersecting, or very close to each other;
Angular tendencies between adjacent DNA linkers;
Interactions between "condensin binding sites" spaced along the string;
Boundary conditions—nucleosomes may not exit a cylindrical or ellipsoidal bounding volume.

In the original implementation, the vast majority (95%) of the runtime is spent computing the repulsion forces that prevent particles from overlapping. As such, the majority of this paper (see Section III) focuses on improving the performance of this kernel, both through algorithmic changes and micro-optimisation. Of secondary importance is the condensin interaction kernel (see Section IV), which consumed 2% of the original simulation runtime. Increasing the dataset size has a greater proportional impact on the performance of this kernel, which therefore is of increasing importance as the code scales.

III. Repulsion force

Two nucleosomes cannot occupy the same area in space at the same time. In order to enforce this in the simulation, a force is included that repels pairs of nucleosomes whose centres come very close to each other. 15 nm is the cut-off radius r_c outside which no repulsion forces apply between two nucleosomes, and is equal to twice the nucleosome radius of 5nm plus an extra 5 nm to deter possible collisions on the next timestep.

The traditional combination of cell lists and Verlet lists are not effective for this calculation, primarily because the timestep Δt is large, and the particles move far enough at each step that rebuilds are necessarily frequent and the lists are large. We now present *projection sorting* as an alternative method and contrast it with Verlet lists in Section III-C.

A. Projection sorting

Given the linear nature of the datasets and the small cut-off radius, we present an alternative approach to cell lists and Verlet lists, henceforth referred to as *projection sorting*. It is easy to see that when two particles are separated by a distance greater than r_c along any single axis (or any unit vector $\hat{v}$), the Euclidean distance between them cannot possibly be less than r_c. This is formalised for two particle position vectors a and b and an arbitrary $\hat{v}$ in Equation 1. It follows that if one were to order the particles by their scalar projection onto such a vector, then for each particle there would exist a contiguous block of particles extending either side within r_c along $\hat{v}$. Only particles within this block could possibly be within r_c in space (subject to a full distance check). Outside of this block all particles could be disregarded. In order for this approach to be effective the span of the set of particles along $\hat{v}$ should greatly exceed that of the span along vectors orthogonal to $\hat{v}$, or many spurious checks must still be carried out.

$$8\hat{v}; a; b\ 2\ \mathrm{R}^3; j(a-b)\ \ \hat{v}j \quad \text{//}a-b\text{//} \tag{1}$$

Our algorithm using this fact is as follows:

1) Selecting $\hat{v}$: An ideal choice for $\hat{v}$ is along a line of best fit through the set of particles, for example, the

ordinary least squares or orthogonal distance regression lines, but this is expensive to calculate. In our case, we have a static, fairly tight, cylindrical/ellipsoidal bounding volume which defines the primary axis along which the string extends. We can reduce the overhead of computing $\hat{v}$ by using the major axis of this volume, which here serves as a good approximation to a line of best fit.

2) Particle sort: The particles are then sorted by their scalar projections onto $\hat{v}$. Gonnet discusses particle sorting by projection in a related context [17]. Efficient sorting is key to good performance with this method.
3) Force sweep: For each particle i, loop over all particles j, where j is bounded by k_{lo} and k_{hi}, the first particles below and above i respectively for which the difference between the scalar projections of j and i onto $\hat{v}$ exceeds r_c. It is guaranteed by Equation 1 that no particle outside this set will be within r_c in space. The search space can be further reduced to particles j, where $i < j < k_{\text{hi}}$ by using Newton's Third Law (N3), at the expense of additional synchronised writes on every inner loop iteration.

B. Implementation

We now discuss the implementation of the two performance-critical components of the projection sorting algorithm—the force sweep and the global particle sort.

The Structure-of-Arrays (SoA) data layout (where each particle facet is laid out independently and contiguously in memory) is used throughout the code for position and force arrays to facilitate vectorisation. As a result, the compiler is able to auto-vectorise the simpler kernels (the entropic, tension and angular forces, and the integration), with minimal help in the form of *#pragma* directives. Those that do not auto-vectorise, including the projection sorting implementation, have been hand-vectorised using both AVX2 and KNC intrinsics (depending on whether the code is being built for CPU or the coprocessor respectively).

1) Force sweep: Due to the nature of vectorisation, all instructions within a branch must be executed if any of the lanes trigger the condition. This introduces inefficiency, as the scalar version only executes the inner branch on a per-particle basis as necessary. The actual inefficiency depends on the proportion of particles that are within the cut-off distance. For a SIMD width of W, up to $W-1$ of the force computations carried out inside the branch could be unnecessary.

Inefficiency also comes from padding to a multiple of the vector width at the end of each force sweep, and from redundant computation due to alignment requirements at the start of each force sweep. Each sweep must continue until all beads in the vector fail the projection cut-off check, which implies a maximum wastage of $2W-2$. The worst case for wastage due to alignment is $W-1$, so for a bidirectional pair of sweeps, the maximum wastage is $6W-6$. The longer the sweep, the smaller a fraction of the total number of particles processed this will account for, leading to better vector efficiency. With real datasets, the sweeps are quite short so the inefficiency can be significant. We explore the empirical values for these inefficiencies in detail in Section III-C.

In addition to inefficiency arising from simply performing unnecessary computation, it is also necessary to ensure that the results of these computations are not stored. This requires 2 extra comparison operations on both the CPU and the coprocessor, 3 extra blend operations on the CPU, and the addition of masking to the final triplet of fused multiply-add instructions used to accumulate forces on the coprocessor.

The force sweep is very cache friendly as all accesses are contiguous. Hardware counter analysis for a representative run reveals that 99.8% of loads issued hit L1 cache. This minimises delays in getting data into the vector registers.

2) Sorting: The other computationally intensive component of the projection sorting approach is the global particle sort. Each thread uses a tuned in-place Quicksort to sort the particles under its control. We can exploit the partially ordered conformation at each step to accelerate the sort somewhat. Pairs of sorted blocks are then merged iteratively using the balanced asynchronous parallel merging algorithm described by Francis and Mathieson [18]. This ensures that each thread merges an even portion of the input sequences. For P threads, $\log_2 P$ layers of merging are required.

The sort operates on the data in SoA format, which is sub-optimal as extraneous data is transferred and takes up cache space, in an already bandwidth intensive operation. Each particle consists of five pieces of information—the value of its scalar projection, its $(x;y;z)$ coordinates in space, and its index in the unsorted conformation. The cost of transposing these five arrays to and from Array-of-Structures (AoS) format (where particle facets are packed in memory) was determined to be significantly more expensive than the overhead incurred by leaving the data as SoA.

Using the SoA format enables the use of vectorised in-register sorting techniques. Bitonic sorting networks [19] are frequently applied here in the literature, as they fit well with existing SIMD ISAs. We use the in-register sorting/merging scheme described by Chhugani et al. [20], implemented with SSE 4.2, AVX2 and KNC intrinsics. Although the size of these networks scales poorly with the SIMD width W we see reasonable speedups of 1.30 , 2.02 and 1.31 for the SSE, AVX2 and KNC implementations respectively, when applying them to sorting a single array. When scaling up to 5 arrays however, the code is much slower than the unvectorised version, peaking at 0.31 . This is due in part to increased register pressure—five times as many arrays requires five times as many registers, and any overflow must be stored on the program stack. However the bigger issue is instruction pressure. The bitonic networks are implemented using shuffle instructions, which for the most part can only be issued to a single execution unit. Analysis using the Intel Architecture Code Analyser (IACA) tool shows huge queues of shuffles lining up against one port, which harms performance greatly as there is no instruction-level parallelism. For this reason we opt not to vectorise the projection sort.

C. Projection sorting vs. Verlet lists

As discussed in Section II, Verlet lists are the *de facto* standard approach to short-range n-body force computation. In this section we investigate how the performance of the projection sorting approach compares. Verlet list rebuilds and force computation using Verlet lists were hand-vectorised as described by Pennycook et al. [11].

To fairly compare the two, we need to choose values for the skin distance r_s and rebuild period k that maximise the performance of the Verlet list approach while still computing correct results. The values $r_s = 40$ and $k = 2$ were determined by tracking the maximum distance moved by any particle over an experiment using the projection sorting method, and setting r_s to just greater than that, ensuring that the results are correct. k was then chosen to maximise performance.

1) Distance check counts: A "distance check" is a calculation of the distance between a pair of particles, necessary to determine whether we need to calculate the force between them. The number of distance checks performed by an algorithm is a good predictor of its performance [21]. Table I shows the average number of distance checks performed using each method, with N3 on and off, as well as the SIMD inefficiency for AVX2 and KNC intrinsic implementations (i.e. the number of distance checks that were unnecessary, and only performed as a result of SIMD limitations).

Projection sorting performs fewer distance checks overall, but is affected more by SIMD inefficiencies. As discussed above, when N3 is not used projection sorting requires two force sweeps. There is SIMD inefficiency at the end of both of these sweeps, and also at the beginning of the sweep due to alignment requirements. Verlet lists only require one sweep regardless of N3, and have no alignment requirements as they are allocated on a cache line boundary. As we scale up to wider SIMD, we see the projection sorting technique approaching the operation of Verlet lists in terms of the number of distance checks performed. At current SIMD widths however, projection sorting still requires the fewest checks in all cases.

2) Verlet rebuild vs. sorting performance: A key part of the Verlet list algorithm is the use of cell lists to accelerate the list build phase. The simulation space is discretised into cubes of side r_v (the Verlet radius, $r_v = r_c + kr_s$) and particles are binned accordingly. While this step is necessary (construction of the Verlet lists takes time quadratic in the number of particles otherwise) the cell lists consume a very large amount of memory. Typically this is avoided by computing the cell lists in a distributed fashion, but in our case we wish to run on a single node, and must find an alternative approach.

As the conformation is concentrated in a small portion of simulation space, we implement the construction using a lock-free hash table, where cell lists are only allocated when a particle actually needs to be added. Once an allocation has occurred we do not free the memory until the end of the simulation, to avoid the large overhead of continually freeing and reallocating memory that is likely to be reused anyway. Using atomic operations rather than mutexes ensures internal consistency with minimal performance penalties. This method is slower than simply allocating all bins at the start, but uses orders of magnitude less memory, and as such is feasible for larger datasets.

Figure 2a compares the costs of Verlet list rebuilds using this scheme and the global particle sort required by the projection sorting algorithm. The sort is clearly cheaper than the Verlet list rebuild, even though it is performed 4 times as often.

3) Sweep performance: Finally, we compare the cost of the force sweeps. Vectorisation is a major consideration, and Table II shows the empirical values for the inefficiency arising from wasted computation inside the force calculation branch. As discussed in Section III-B, we see very high fractions approaching $\frac{W-1}{W}$ here due to the low rate of interactions between particles (brought on by the small cut-off distance). This impacts the overall SIMD speedup as the width increases. Verlet lists have slightly lower inefficiencies as they preserve the order of beads better than projection sorting.

Figure 2b shows the full sweep comparison. Interestingly, the fastest option here is projection sorting with N3 *disabled*. Even though N3 cuts the number of distance checks in half, the additional cost of atomic operations on the force array outweigh this benefit. The gap is especially pronounced on Xeon Phi, as it is running 15 as many threads. Conversely, N3 improves performance for Verlet lists.

In conclusion, projection sorting wins on all fronts in these tests, exhibiting the lowest number of distance checks, the cheapest periodic costs, and the fastest force sweeps. The high value for r_s is the primary reason that Verlet lists are ineffective for this simulation, as this forces the rebuild period lower, and increases the list sizes. Nonetheless it is clear that, projection sorting can be an effective alternative. The primary factors to consider when choosing an algorithm are (roughly in order of importance):

Geometry of the simulation (the set of particles having a long axis favours projection sorting),
Average movement of particles per timestep (lower allows for a smaller r_s),
Projection sorting uses memory bandwidth more effectively,
Higher SIMD width favours Verlet lists.

IV. Condensin force

The other computationally intensive force calculation pertains to the interactions between "condensin binding sites"—modelled as special nucleosomes occurring along the length of the string at irregular intervals, with an average separation of 48 nucleosomes. These sites can interact when they come close, and become stuck together for extended periods, prompting the condensation of the string over time.

Sites whose centres come within 40 nm of each other experience attractive forces, up to a limited number of interactions per site, per timestep (typically capped at 1 or 2). There is also a stochastic component—for each interaction, and each timestep there is a small configurable probability that interacting sites will dissociate from each other. When this

Algorithm	N3?	Mean # checks	Mean AVX2 ineff. (#/%)	Mean KNC ineff. (#/%)
Projection sorting	N	64.73	10.15 (13.55%)	22.03 (25.39%)
	Y	32.41	6.00 (15.54%)	13.98 (30.13%)
Verlet lists	N	91.74	1.42 (1.52%)	3.54 (3.72%)
	Y	45.83	1.57 (3.31%)	3.60 (7.27%)

TABLE I: Mean number of distance checks performed per particle, and the number of unnecessary checks performed as a result of SIMD inefficiencies for AVX2 (4-wide) and KNC (8-wide) implementations. The dataset used contained 128,000 nucleosomes, with $r_s = 40$ and $k = 2$.

Algorithm	N3?	Mean # calcs.	Mean AVX2 ineff. (#/%)	Mean KNC ineff. (#/%)
Projection sorting	N	1.68	4.74 (73.83%)	10.51 (86.21%)
	Y	0.84	2.38 (73.91%)	5.47 (86.69%)
Verlet lists	N	1.68	3.53 (67.75%)	7.80 (82.28%)
	Y	0.84	2.05 (70.93%)	4.69 (84.81%)

TABLE II: Mean number of full neighbour force calculations performed per particle, and the number of unnecessary calculations performed as a result of SIMD inefficiencies for AVX2 (4-wide) and KNC (8-wide) implementations. The dataset used contained 128,000 nucleosomes, with $r_s = 40$ and $k = 2$.

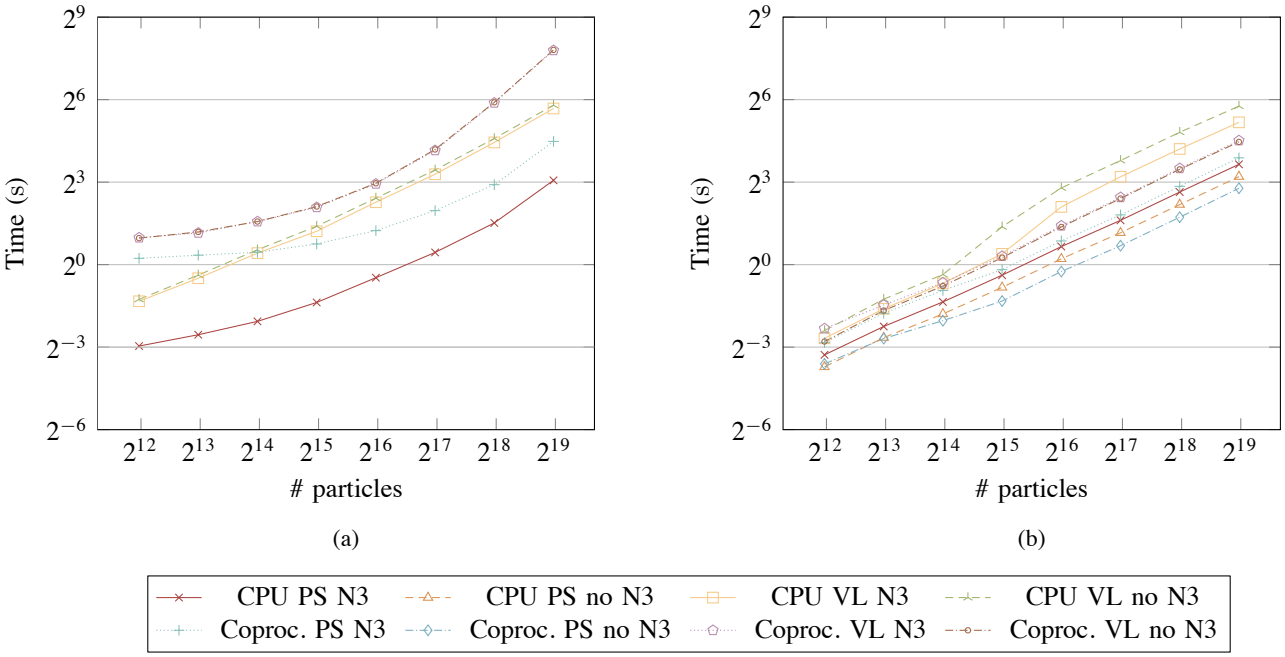

Fig. 2: Breakdown of performance differences between: (a) Verlet list rebuilds vs. projection sorts, and (b) force sweep based on Verlet lists and force sweep based on projection sorting, over different dataset sizes. The CPU is running 16 threads and the coprocessor 244. Only the "N3" lines are shown for projection sorting in (a), as this setting doesn't affect the sort.

happens they enter a cooldown period of 3 timesteps during which they cannot form any bonds, giving them time to move apart.

There are two primary steps to computing the forces on each site, referred to henceforth as the binning step and the interaction step respectively:

1) List other sites within the 40 nm cut-off radius,
2) Determine whether to apply forces, dissociate, or advance cooldown period, depending on the number of close pairs and their interaction history.

This is another short-range interaction of the type discussed in Section III, although with different properties. The cut-off radius is larger, 40nm compared to 15nm. The force between a pair of sites is more expensive to calculate, but the number of sites is an order of magnitude smaller than the number of nucleosomes. For a given site, we must determine all other close sites before we can compute any forces, rather than accumulating them per interaction as in the repulsion kernel. The cooldown mechanic also introduces additional state between timesteps, which complicates matters. Projection

sorting can be used during the binning step, although careful attention must be paid to correctly mapping from sorted binding sites to the inter-timestep state.

A. Implementation

1) Storage: It is necessary to store the cooldown status for every pair of binding sites—a flag indicating whether a site's interaction with another bead is currently in cooldown mode, and the number of timesteps remaining before it is free to interact again. While the two can be combined into a single field (with 0 representing no cooldown mode, and any other number representing the remaining count), the naïve storage requirement is still quadratic in the number of sites. This becomes a problem with larger datasets.

As dissociation events are uncommon, the matrix of cooldown state is very sparse. Taking advantage of this fact, we implement the same technique used to reduce the storage requirement for cell lists in Section III-C, and replace the matrix with a lock-free hash table. For a large number of binding sites, say 100,000, this approach requires over 2300 less space, 4.1 MB instead of 9.3 GB.

2) Vectorisation: Meaningful vectorisation is infeasible for both the binning and interaction steps. The binning step requires access to the cooldown status of each bead. As these are stored non-contiguously regardless of the storage strategy used, we are faced with an expensive gather operation. In the case of the hash table, current SIMD ISAs do not support atomic gathers [22], necessitating performing the memory accesses and register insertions manually. More crucially though, the average binding site sweep length is slightly under 2, which negates any benefit due to the large overhead. For the interaction step the algorithm dictates that each site is processed individually based on the contents of a very short list of neighbour sites.

V. Results

We now discuss in more detail the experimental setup for the runs performed, and present both the overall runtime characteristics and comparisons between performance on the CPU and coprocessor.

A. Experimental setup

All experiments use the projection sorting method without N3, as discussed in Section III-C. Per kernel timing was implemented using the `rdtsc` hardware counter in order to achieve high accuracy with minimal overhead.

1) Datasets: The initial dataset described by Cheng et al. [1] was derived from a budding yeast cell and contains 2000 nucleosomes. As no larger real datasets were available while this work was being undertaken, we generated extended versions of the original using probabilistic methods. We defined three normal distributions, each parameterised using the mean and standard deviation of the deltas between each nucleosome for the x, y and z axes respectively. We then generated new conformations of length N particles by sampling these distributions to perform an N step random walk. Condensin binding sites were placed randomly on average every 48 nucleosomes. After generation we advanced the conformation by 100,000 timesteps to reach a relatively stable state, free of artefacts caused by the random walk process. Synthetic datasets were generated for the following values of N: 4000, 8000, 16,000, 32,000, 64,000, 128,000, 256,000 and 512,000.

	Xeon E5-2630v3	Xeon Phi 7120P
Sockets Cores Threads	2 8 2	1 61 4
Clock (GHz)	2.40	1.24
L1fi,dg / L2 / L3 Cache (KB)	32 / 256 / 12288	32 / 512 / N/A
Memory (GB)	64	16
SIMD ISA	AVX2	KNC

TABLE III: Machine configuration

2) Machine specifications: The machine used for experiments was fitted with dual Intel Xeon E5-2630 8-core CPUs for a total of 16 cores. 64GB of RAM was available. The coprocessor was an Intel Xeon Phi 7120P, with 61 cores and 16GB of RAM. See Table III for details. All code was compiled using the Intel C++ compiler, v15.0.4.

B. Overall performance

Relative to the original code, we see single-threaded speedups starting at over 10 on the CPU for 2000 beads (see Figure 4 for a per-kernel breakdown), and increasing as the dataset size goes up due to better algorithmic scaling. We would note that this is not a fair comparison of algorithmic approaches (previously presented in Section III-C), as the original cell list implementation is not heavily optimised. The slowdown to the entropic kernel is due to switching to a more robustly thread-safe random number generator. We observe speedups in all other kernels. This decreases the time taken to perform a typical experimental run, consisting of 40 million timesteps, from 90 hours to 9 hours on our hardware. When factoring in the effects of parallelisation the improvement is much greater.

Figure 3 shows a breakdown of each optimised kernel's performance over a range of dataset sizes for both the CPU and coprocessor. On the CPU, the repulsion sweep is the most expensive, followed closely by the condensin interactions and the sort. The entropic, tension and attraction forces (grouped under "other") are comparatively cheap. The point where the integration falls out of last-level cache (LLC) is clearly visible between 128,000 and 256,000. The barrier costs are fairly low throughout, but increase sharply for the largest dataset, possibly due to non-uniform memory access (NUMA) problems.

On the coprocessor, the sort is most expensive, primarily as it is not vectorised at all (see discussion in Section III-B). Vectorisation is more crucial to performance on the Xeon Phi than on the CPU so this is expected. The repulsion sweep is cheaper on the Xeon Phi, as it vectorises very well and does not require any barriers. Interestingly, the condensin interactions are also cheaper to compute, despite not being

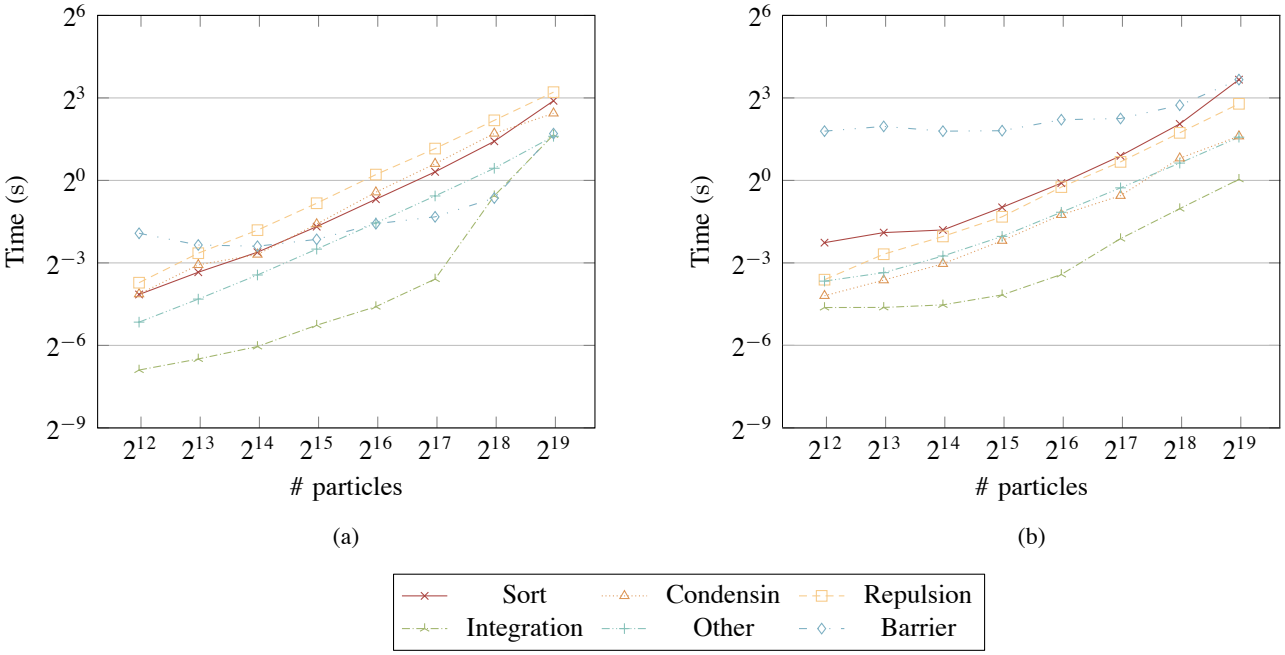

Fig. 3: Breakdown of kernel times when running on the CPU and the coprocessor across a range of dataset sizes. (a) shows the timings for 16 threads on the CPU, (b) shows 244 threads running on the coprocessor.

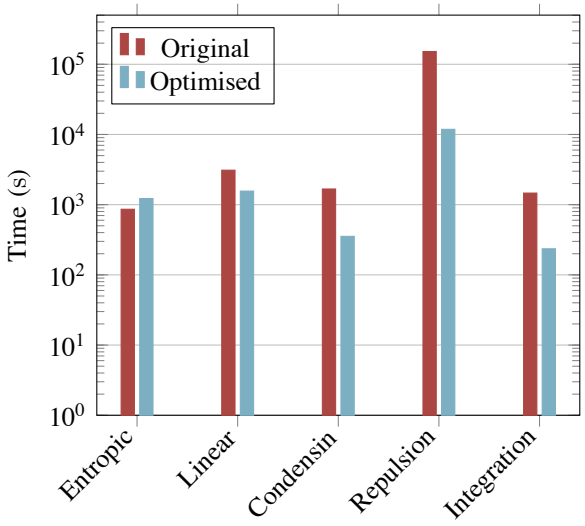

Fig. 4: Per-kernel comparison of single-threaded performance between the original application and our optimised version for the 2000 nucleosome yeast dataset. The "linear" kernel refers to the combination of the tension and angular force computation, which were merged for the optimised version. A logarithmic y-axis is used to better demonstrate the difference in terms of orders of magnitude—note the 10 improvement to the repulsion kernel.

vectorised either, likely because each binding site is largely independent leading to good scaling to a larger number of threads. The integration also scales better with dataset size, as the coprocessor has roughly 3 the memory bandwidth as the CPU (153 GB=s per NUMA region as opposed to 48 GB=s, as reported by the STREAM benchmark [23]).

The main issue we see on the coprocessor is significantly higher barrier costs. On some level this is unavoidable, a higher number of threads is going to mean slower blocking operations and a greater sensitivity to load imbalance, and we cannot remove any barriers as they are necessary to ensure correctness. We can aim to reduce the number of barriers via algorithmic changes however—the midpoint integration scheme used is the main culprit here, requiring twice as many barriers per timestep as would otherwise be needed.

C. Offload computation

We experimented with offloading computation to the coprocessor while running on the CPU. Suitable candidate kernels for offloading should perform well on the coprocessor, be able to run in parallel with other kernels (minimal data dependencies), not require large amounts of data transfer on and off the coprocessor each timestep, and take long enough that the overhead of offload does not dominate. Of the kernels in this simulation, the only one that satisfies most of these conditions is the projection sorting force sweep. It performs better on the coprocessor, and can be run in parallel with any of the other force computation kernels. Despite this, the time saved by running offloaded was roughly equalled by the overhead of doing so, and we did not see any significant change in performance.

VI. Conclusions

We present *projection sorting*, an alternative to the traditional Verlet list algorithm for short-range interaction force computation, and show that it is more effective under certain conditions

present in this molecular dynamics simulation of chromosome condensation. We provide efficient parallel implementations of this strategy for traditional and many-core architectures, along with the rest of the code.

We achieve large speed-ups starting at 10 , and improving with dataset size, over the original implementation, and compare the performance of our optimised CPU and coprocessor implementations. We find that some kernels are better suited to the Xeon Phi coprocessor, in particular the projection sorting force sweep, which consumes the majority of the runtime in this simulation.

Our optimisations have been and continue to be used to facilitate further experiments into chromosome condensation. While the algorithms we discuss are specific to molecular dynamics, the issues that arise through implementation are more widely applicable, in particular our discussion of sorting, a very common operation in a great many classes of code.

A. Further work

Future directions for this code include support for multiple interacting chromatin strings with a controlled region of overlap between their bounding boxes, which introduces some challenging dynamic load balancing problems where the strings come into contact with each other.

Reworking the simulation to use an alternative integration scheme that does not require computing forces more than once (such as Verlet integration [14]), would radically affect the performance characteristics of the simulation.

Currently the code is shared memory parallel only, using OpenMP. Extension to distributed memory parallelism would require reworking of some algorithms (for example, efficient distributed sorting is more complex than shared memory sorting [24]), but is generally straightforward and would open doors to greater performance on larger systems.

ACKNOWLEDGEMENTS

This work was supported by the Francis Crick Institute which receives its core funding from Cancer Research UK (FC001003), the UK Medical Research Council (FC001003), and the Wellcome Trust (FC001003), and by the Engineering and Physical Sciences Research Council and Intel Corporation (CASE award 1365607).

REFERENCES

[1] T. M. K. Cheng, S. Heeger, R. A. G. Chaleil, N. Matthews, A. Stewart, J. Wright, C. Lim, P. A. Bates, and F. Uhlmann, "A simple biophysical model emulates budding yeast chromosome condensation." *eLife*, vol. 4, p. e05565, 2015.

[2] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten, "Scalable molecular dynamics with NAMD." *Journal of Computational Chemistry*, vol. 26, no. 16, pp. 1781–1802, Dec. 2005.

[3] A. Bhatele, S. Kumar, C. Mei, J. C. Phillips, G. Zheng, and L. V. Kalé, "Overcoming scaling challenges in biomolecular simulations across multiple platforms," in *Proceedings of the International Parallel and Distributed Processing Symposium 2008*. IEEE, 2008, pp. 1–12.

[4] W. Jiang, J. C. Phillips, L. Huang, M. Fajer, Y. Meng, J. C. Gumbart, Y. Luo, K. Schulten, and B. Roux, "Generalized Scalable Multiple Copy Algorithms for Molecular Dynamics Simulations in NAMD." *Computer Physics Communications*, vol. 185, no. 3, pp. 908–916, Mar. 2014.

[5] Y. Sun, G. Zheng, C. Mei, E. J. Bohm, J. C. Phillips, L. V. Kalé, and T. R. Jones, "Optimizing fine-grained communication in a biomolecular simulation application on Cray XK6," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis 2012*. IEEE, 2012, pp. 1–11.

[6] S. J. Plimpton, "Fast Parallel Algorithms for Short-Range Molecular Dynamics," *Journal of Computational Physics*, vol. 117, pp. 1–19, Mar. 1995.

[7] I. T. Todorov, W. Smith, K. Trachenko, and M. T. Dove, "DL_POLY_3: new dimensions in molecular dynamics simulations via massive parallelism," *Journal of Materials Chemistry*, vol. 16, no. 20, pp. 1911–1918, May 2006.

[8] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl, "GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers," *SoftwareX*, vol. 1-2, pp. 19–25, Sep. 2015.

[9] J. A. Anderson, C. D. Lorenz, and A. Travesset, "General purpose molecular dynamics simulations fully implemented on graphics processing units," *Journal of Computational Physics*, vol. 227, no. 10, pp. 5342–5359, May 2008.

[10] A. Harode, A. Gupta, B. Mathew, and N. Rai, "Optimization of Molecular Dynamics application for Intel Xeon Phi coprocessor," in *Proceedings of the International Conference on High Performance Computing and Applications 2014*. IEEE, 2014, pp. 1–6.

[11] S. J. Pennycook, C. J. Hughes, M. Smelyanskiy, and S. A. Jarvis, "Exploring SIMD for Molecular Dynamics, Using Intel R Xeon R Processors and Intel R Xeon Phi^{TM} Coprocessors," in *Proceedings of the International Symposium on Parallel and Distributed Processing 2013*. IEEE Computer Society, May 2013, pp. 1085–1097.

[12] R. W. Hockney, S. P. Goel, and J. W. Eastwood, "Quiet high-resolution computer models of a plasma," *Journal of Computational Physics*, vol. 14, no. 2, pp. 148–158, Feb. 1974.

[13] B. Quentrec and C. Brot, "New method for searching for neighbors in molecular dynamics computations," *Journal of Computational Physics*, vol. 13, no. 3, pp. 430–432, Nov. 1973.

[14] L. Verlet, "Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules," *Physical Review*, vol. 159, no. 1, pp. 98–103, Jul. 1967.

[15] Z. Yao, J. Wang, G. Liu, and M. Cheng, "Improved neighbor list algorithm in molecular simulations using cell decomposition and data sorting method," *Computer Physics Communications*, vol. 161, no. 1-2, pp. 27–35, Aug. 2004.

[16] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz, "Analysis of the clustering properties of the Hilbert space-filling curve," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 1, pp. 124–141, 2001.

[17] P. Gonnet, "A simple algorithm to accelerate the computation of non-bonded interactions in cell-based molecular dynamics simulations." *Journal of Computational Chemistry*, vol. 28, no. 2, pp. 570–573, Jan. 2007.

[18] R. S. Francis and I. D. Mathieson, "A benchmark parallel sort for shared memory multiprocessors," *IEEE Transactions on Computers*, vol. 37, no. 12, pp. 1619–1626, 1988.

[19] K. E. Batcher, "Sorting networks and their applications," in *Proceedings of the 1968 AFIPS Conference*. ACM Press, 1968, pp. 307–314.

[20] J. Chhugani, A. D. Nguyen, V. W. Lee, W. Macy, M. Hagog, Y. Chen, A. Baransi, S. Kumar, and P. Dubey, "Efficient implementation of sorting on multi-core SIMD CPU architecture," in *Proceedings of the VLDB Endowment 2008*. VLDB Endowment, Aug. 2008, pp. 1313–1324.

[21] U. Welling and G. Germano, "Efficiency of linked cell algorithms," *Computer Physics Communications*, vol. 182, no. 3, pp. 611–615, Mar. 2011.

[22] S. Kumar, D. Kim, M. Smelyanskiy, Y. Chen, J. Chhugani, C. J. Hughes, C. Kim, V. W. Lee, and A. D. Nguyen, "Atomic Vector Operations on Chip Multiprocessors," in *Proceedings of the 35th International Symposium on Computer Architecture 2008*. IEEE, 2008, pp. 441–452.

[23] J. D. McCalpin, "Memory Bandwidth and Machine Balance in Current High Performance Computers," *IEEE Computer Society Technical Committee on Computer Architecture TCCA Newsletter*, 1995.

[24] E. Solomonik and L. V. Kalé, "Highly scalable parallel sorting," in *Proceedings of the International Parallel and Distributed Processing Symposium 2010*. IEEE, 2010, pp. 1–12.